The Dynamic And-Or Quorum System

Uri Nadav[1] and Moni Naor[2]

[1] Dept. of Computer Science Tel-Aviv University
[2] Dept. of Computer Science and Applied Mathematics, The Weizmann Institute

Abstract. We investigate issues related to the probe complexity of the And-Or quorum system and its implementation in a dynamic environment. Our contribution is twofold: We first analyze the algorithmic probe complexity of the And-Or quorum system, and present two optimal algorithms. The first is a non-adaptive algorithm with $O(\sqrt{n}\log n)$ probe complexity, which matches a known lower bound. The second is an adaptive algorithm with a probe complexity that is linear in the cardinality of a quorum set ($O(\sqrt{n})$), and requires at most $O(\log\log n)$ rounds. To the best of our knowledge, all other adaptive algorithms with same parameters (load and probe complexity) require $\theta(\sqrt{n})$ rounds.

Our second contribution is presenting the *'dynamic And-Or'* quorum system - an adaptation of the above quorum system to a dynamic environment, where processors join and leave the network. It is based on a dynamic overlay network that emulates the De-Bruijn network and maintains the good properties of the quorum system(e.g.,load and availability). The algorithms suggested for the maintenance of these dynamic data structures are strongly coupled with the dynamic overlay network. This fact enables the use of gossip protocols which saves in message complexity and keeps the protocols simple and local. All these qualities make the *'dynamic And-Or'* an excellent candidate for an implementation of dynamic quorums.

1 Introduction and Motivation

Quorum systems (QS) serve as a basic tool providing a uniform and reliable way to achieve coordination between processors in a distributed system. Quorum systems are defined as follows: A set system is a collection of sets $\mathcal{S} = \{S_1, \ldots, S_m\}$ over an underlying universe $U = \{u_1, \ldots, u_n\}$. A set system is said to satisfy the *intersection property*, if every two sets $S, R \in \mathcal{S}$ have a nonempty intersection. Set systems with the intersection property are known as *quorum systems* and the sets in such a system are called quorums.

We are interested in two aspects of quorum systems that arise from peer-to-peer environments, where processors are loosely coupled and may join and leave the network at will:

1. The setting in which the quorum operates is often dynamic, and should accommodate changes in the quorum system over time. We address the problem of designing a quorum system that is fit for a scalable and dynamic environment where processors leave and join at will.

P. Fraigniaud (Ed.): DISC 2005, LNCS 3724, pp. 472–486, 2005.

2. In a dynamic setting a global view of the system is usually hard to maintain and is therefore not assumed. We are interested in solutions which use local algorithms for the maintenance of the QS i.e., ones in which the input for a processor only comes from its local view of its neighbors in the network.

This paper deals with the And-Or quorum system, presented by Naor and Wool[16]. We examine the And-Or system both in a static environment where the set of processors is fixed, and in a dynamic and scalable environment. We conclude that it is very suitable for implementation in a dynamic setting.

1.1 Definitions and Preliminaries

The *load* on a system of processors, as defined in [16], captures the probability of accessing the busiest processor. An *access strategy* μ, for a quorum system $\mathcal{Q}$, is a probability distribution over the quorums in $\mathcal{Q}$, i.e. $\sum_{Q \in \mathcal{Q}} \mu(Q) = 1$. An access strategy can be thought of as a method for choosing a quorum. The load $\mathcal{L}(i)$ induced by a strategy on a processor $i \in U$ is the probability that processor i is chosen by the strategy, i.e. $\mathcal{L}(i) = \Pr_\mu[i \in Q]$. The load of a strategy μ is $\mathcal{L}(\mu) = \max_{i \in U} \mathcal{L}(i)$. For a quorum system $\mathcal{Q}$, the load $\mathcal{L}(\mathcal{Q})$ is the minimum load induced by a strategy, over all strategies. Let $c(\mathcal{Q})$ denote the size of the smallest quorum in $\mathcal{Q}$. Naor and Wool prove in [16] the following lemma:

Lemma 1 ([16]). *The load of a quorum system $\mathcal{Q}$ is at least $max\{\frac{1}{c(\mathcal{Q})}, \frac{c(\mathcal{Q})}{n}\}$ which implies that $\mathcal{L}(\mathcal{Q}) \geq \frac{1}{\sqrt{n}}$.*

We assume that processors may temporarily fail(halt). The failure of a processor occurs with some fixed probability p and is independent from failures of other processors in the network (Random faults model). A quorum is said to be alive if all of its elements are alive (non-faulty), and dead otherwise. The *availability* of a quorum system $\mathcal{Q}$ is measured by the global failure probability F_p which is the probability that all quorums in $\mathcal{Q}$ are dead. This failure probability measures how resilient the system is.

The availability measure tells us about the probability of a quorum set to survive in the random-fault model, but tells us nothing about the complexity of finding a live quorum. The probe complexity of a quorum system captures the number of probes required for finding a live quorum (cf. [5],[18]). We use the same probing model defined in [18], where a user learns the state of a processor with a single probe (by sending a message and waiting a timeout period for the reply) i.e., faults are detectable. It is also assumed that the state of a processor does not change while the system is being probed. A lower bound on the algorithmic probe complexity of a non-adaptive algorithm, as a function of the load was given in [15]:

Theorem 1 ([15]). *Let $\mathcal{Q}$ be a quorum system over a universe of processors U with load $\mathcal{L} = \mathcal{L}(\mathcal{Q})$. Assume that each element in U fails with some fixed probability $p \leq \frac{1}{2}$. Let $X \subset U$ be a predefined set of elements (that corresponds to a non-adaptive algorithm) s.t.,*

$$\Pr[X \textit{ contains a live quorum}] \geq \frac{1}{2},$$

then

$$|X| \geq \frac{1}{2\log(1/p)+1}\frac{\log(1/4\mathcal{L})}{\mathcal{L}}.$$

In particular if $\mathcal{L} = O(\frac{1}{\sqrt{n}})$ *then* $|X| = \Omega(\sqrt{n}\log n)$.

A *non-adaptive* algorithm for finding a live quorum decides on which processors to probe prior to gaining any knowledge as to which processors failed. Such algorithms are attractive since they could be executed in parallel. An *adaptive* algorithm on the other hand, can act in a number of rounds, such that in each round the probe results of prior rounds are known.

1.2 Quorum System for P2P and Dynamic Environments

Recently, peer-to-peer applications gained enormous popularity, which led to an extensive research on dynamic distributed data structures on overlay networks. Most of the attention was put on dynamic hash tables (c.f [10][14][19][20][21]). In these constructions data items are divided locally and on the fly between active processors in the network. An overlay network is built up dynamically, as processors join and leave the network.

The design of quorum systems for a dynamic environment raises two problems:

Integrity: When processors join or depart the system the quorum sets should be modified. The integrity of the system is preserved in two aspects: First, the intersection property must hold. Second, the adaptation should keep the properties of the quorum system e.g., load and availability.

Locality: Even if the set of quorums is well defined, it is required of a user to actually know the quorum structure in order to choose a quorum. In a dynamic system a global view of the system is hard to maintain and is therefore not assumed. The users of the quorum system are the processors themselves. When a processor selects a quorum it has no knowledge of the complete quorum structure and only has access to its close neighborhood.

We combine techniques from works on dynamic overlay networks together with the And-Or quorum system and come up with a design that provides high scalability and locality while preserving other measurements of quality from classic research of quorum systems.

1.3 Related Work

Peleg and Wool defined the availability of quorum systems in [17] and studied the availability of some quorum system classes. In [18] Peleg and Wool studied the complexity of finding a live quorum or evidence for the lack of, in an adversarial-fault model. In this model crash faults are assumed to be detectable, and a probe

action to a processor is assumed to have $O(1)$ complexity. In [5] Hassin and Peleg analyzed the probe complexity of several systems in the random-fault model. Bazzi [3] introduced the term *'cost of failures'*. Given a network implementation and an algorithm for finding a live quorum, the cost of failures measures the average communication overhead caused by encountering a faulty processor. In [15] Naor and Wieder defined the *Algorithmic probe complexity* as the actual time and message complexity required for finding a live quorum.

In [16] Naor and Wool presented both the *Paths* quorum system and the *And-Or* system, and showed that they both have asymptotically optimal load and availability. In [15], Naor and Wieder suggested an adaptation of *Paths* in a dynamic environment and showed that it maintains the optimal load and availability of the original system. Malkhi et al [11] suggested a relaxation for quorum systems in which the intersection of two quorums is only required with some high probability. It was shown that such a relaxation results in a dramatic improvement in the resilience of the system. Abraham and Malkhi [2] introduced the *'dynamic ϵ-intersecting system* which is an adaptation of the ϵ-intersecting system for a dynamic environment.

In [13] we have shown an embedding of the *'And-Or tree'* structure over the *'Distance-Halving'* overlay network [14], which served as a fault-tolerant storage system. We use the same embedding here for the construction of a dynamic quorum system. In quorum systems the major design goal is to ensure intersection between quorums, whereas in storage systems, files reconstruction serves as the main target. The intersection property of quorum-systems is necessary but not sufficient for this matter. We wish to emphasize this difference in goals which leads to entirely different algorithms and analyses. We also emphasize the fact that the fault-model in [13] is adversarial, in contrast with a random-fault model analyzed in this paper.

1.4 Results and Paper Organization

This paper is divided into two parts. In the first part, we analyze the algorithmic probe complexity of the *And-Or* quorum system (Section 2). We show a non-adaptive algorithm for finding a live quorum with $O(\sqrt{n}\log n)$ probe complexity. This matches a lower bound of Naor and Wieder [15]. We further show an adaptive algorithm with a probe complexity of $O(\sqrt{n})$, which is optimal (up to constants). Thus combined with the results in [16] the *And-Or* system is shown to have an excellent balance between many somewhat contradictory measures of quality.

The *'Paths'* quorum system was already shown to achieve optimal load, availability [16], and algorithmic probe complexity [15]. The adaptive algorithm for paths requires $O(\sqrt{n})$ probes as in our case. However, it requires $\theta(\sqrt{n})$ rounds, whereas our algorithm only requires $O(\log\log n)$ rounds. To the best of our knowledge, no other algorithm with similar parameters works as fast. Our algorithm also achieves early stopping in the case of no faults. Our contribution is therefore twofold: First, our adaptive algorithm works very fast. Second, the analysis is kept simple in contrast with the results in [16][15] which require heavy tools from percolation theory.

We should note however that the *Paths* quorum system has a better availability than And-Or, in the sense that it remains available for every failure probability $p < \frac{1}{2}$. In contrast, it was shown in [16] that the And-Or has the same asymptotic behavior, but for $p < p_c$, where p_c is a constant strictly less than $\frac{1}{2}$.

In the second part of this paper (Section 4) we present and analyze *'dynamic And-Or'*, a construction for a dynamic and scalable quorum system which could be viewed as a dynamic adaptation of the And-Or system. We show that the dynamic And-Or keeps the optimal load, availability and probe complexity of the And-Or system.

The remainder of this paper is organized in the following way. In Section 2 we present the optimal adaptive and non-adaptive algorithms for finding a live quorum in the random-fault model. Section 3 analyzes the And-Or quorum system on a balanced binary tree which is incomplete. This structure is a step towards the adaptation of the And-Or for a dynamic environment, which is presented in Section 4. Section 4 then presents two different implementations of the And-Or system in a dynamic environment. Finally, our conclusions and open questions are presented in Section 5. We note that all proofs are omitted from this version due to lack of space.

2 The And-Or Quorum System

We recall the construction of the *'And-Or'* quorum system presented in [16]. Consider a complete binary tree of height h, rooted at *root*, and identify the 2^h leaves of the tree with systems processors. We define two collections of subsets over the set of processors, using the following recursive definitions:

(i) For a leaf v, $\text{ANDset}(v) = \text{ORset}(v) = \big\{\{v\}\big\}$.
(ii) $\text{ANDset}(v) = \big\{S \cup R | S \in \text{ORset}(v.left) \wedge R \in \text{ORset}(v.right)\big\}$.
(iii) $\text{ORset}(v) = \text{ANDset}(v.left) \cup \text{ANDset}(v.right)$

The And-Or quorum system is then composed of the collections ANDset (*root*) and ORset(*root*). A natural recursive procedure can be used to generate a set $S \in \text{ANDset}(root)$. The procedure visits nodes of the tree, beginning at the root and propagating downwards. It considers the nodes of the tree as AND/OR gates (on even/odd levels). When visiting an AND gate it continues to both its children, and when visiting an OR gate one of its children is chosen. The leaves visited by the procedure form S. A similar procedure generates a set $R \in \text{ORset}(root)$. The And-Or structure induces the following properties:

Lemma 2 (from [16]). *Consider a complete binary tree of height h rooted at root. Let $S \in \text{ANDset}(root)$ and $R \in \text{ORset}(root)$ then $|S \cap R| = 1$, $|R| = 2^{\lfloor \frac{h}{2} \rfloor}$ and $|S| = 2^{\lfloor \frac{h+1}{2} \rfloor}$.*

Lemma 2 states that each set $S \in \text{ANDset}(root)$ intersects each set $R \in \text{ORset}(root)$. This property is used to define the *'And-Or'* quorum system: A

quorum in the And-Or system is any set $Q = S \cup R$ where S is a member of ANDset($root$) and R is a member of ORset($root$). For a binary tree T we denote the set of quorums with AndOr(T). The load of the And-Or system was shown to be optimal ($O(\frac{1}{\sqrt{n}})$) in [16].

The And-Or system was shown to have a failure probability F_p which exponentially decays in n. For $p \leq \frac{1}{4}$ it was shown in [16] that $F_p(\text{AndOr}) \leq \exp(-\Omega(\sqrt{n}))$.

2.1 A Non-adaptive Algorithm

We now show a non-adaptive algorithm for choosing a live quorum in the And-Or quorum system. The probe complexity of this algorithm is $O(\sqrt{n}\log n)$ which matches the lower bound of Theorem 1.

Let r be the root of a complete binary tree. We introduce the notion of ANDset collection on internal nodes of the tree, in level h, denoted ANDset(r, h):

Definition 1 (ANDset(r, h)). *Let T be a binary tree rooted at r with h levels. Let T' be the same tree, where all the nodes in level $h+1$ or above are truncated, and the nodes of level h serve as leaves. Then* ANDset(r, h) *equals the collection* ANDset(r) *as defined on T'.*

The non-adaptive algorithm uniformly picks a set $R \in \text{ANDset}(r, h)$ where

$$h = \lfloor \log n - 2 \log \log n \rfloor.$$

All processors which are descendants of nodes in R are then chosen. The cardinality of such a set follows from Lemma 2.

Lemma 3. *The cardinality of a set of processors S chosen by the non-adaptive algorithm is $O(\sqrt{n}\log n)$.*

Observation 1 follows directly from the definition of the recursive procedure for selecting a member of ANDset(r):

Observation 1. *Let S be a member of* ANDset(r) *then there exists a set $R \in$* ANDset(r, h) *and a collection $\{S_v : v \in R, S_v \in \text{ANDset}(v)\}$ such that:*

$$S = \cup_{v \in R} S_v.$$

From the high availability property of the And-Or, and from Observation 1 it follows that a live member of ANDset(r) is contained in the probed set, with high probability[1] .

Theorem 2 (Non-adaptive Probe Complexity). *Let S denote the set of processors chosen as described above, then S contains a live set from the collection* ANDset(r) *w.h.p .*

In a similar way a subset of processors that contains a live member of ORset(r) can be chosen. Joining such a set with a live member of ANDset(r) yields a live quorum in the And-Or quorum system.

[1] By "with high probability" (w.h.p.), we mean "with probability at least $1 - O(n^{-\alpha})$ for some constant $\alpha > 0$, for a system with n processors."

2.2 An Adaptive Algorithm

We present an adaptive algorithm for finding a live quorum for the And-Or system which requires $\theta(\sqrt{n})$ probes. By an adaptive algorithm we mean an algorithm that probes several subsets of processors, in a number of rounds. The algorithm is assumed to send all messages (probes) in a specific round prior to getting any reply from any of the processors that are probed in this round. Answers from probed processors are returned before some timeout, after which the algorithm decides which processors to probe in the next round. The number of rounds therefore determines the time complexity of the algorithm so it is desired to have as small as possible number of rounds.

Our algorithm requires w.h.p. only $O(\log \log n)$ rounds. Naor and Wieder presented in [15] an adaptive algorithm for the Paths quorum system with a $\theta(\sqrt{n})$ probe complexity, however, their solution requires $\theta(\sqrt{n})$ rounds.

As in the non-adaptive case, it is sufficient to show how to find a live member of ANDset($root$). A live member of ORset($root$) can be found in a similar way. To adaptively find a live member of ANDset($root$), the algorithm first picks an arbitrary set $S \in$ ANDset($root$) and probe its members. If S turns out to be alive then the algorithm may finish. Otherwise, the set is being 'corrected', by replacing the choices that turned out to be *dead*, in the following way.

Let parent(v) denote the internal node in the tree which is the parent of v. To correct a choice of a *dead* processor $v \in S$, the leaves in the subtree $v' =$ parent(v) are probed. If the newly probed processors contain a live member of ANDset(v') then the algorithm can finish. Otherwise, the last step is recursively repeated with parent(v'). This process runs in parallel for each faulty processor $v \in S$. Following from Observation 1, when all processes stop, the probed elements contain a live member of ANDset(r). If a live member exists, then by the time all the process reach the root of the tree, this live member is found. Algorithm 1 describes this process more formally.

We turn to the analysis of the probe complexity of the adaptive algorithm. Step 2 of Algorithm 1 finds for each $v \in S$ its lowest ancestor v', for which a live member in ANDset(v') exists. Note that if v is alive then $v' = v$. Denote

Algorithm 1 Adaptive Find Live Quorum

Input: Oracle access to the state of processors (*live* or *dead*).

Output:A live member of ANDset(r) (or *'fail'* in case none was found).

1. Probe the members of some set $S \in$ ANDset($root$).
2. For each $v \in S$ which is *dead*, do the following:
   (a) Set $v' =$ parent(v)
   (b) Probe all leaves of v'
   (c) While the leaves in the subtree v' do not contain a live member of ANDset(v'), set $v' \leftarrow$ parent(v') and go back to stage 2b.
   (d) Add the members of the live set to S.

by h_v the distance from v to this v'. h_v determines the number of rounds for each process. The number of rounds of the algorithm is $max_{v \in S}\{h_v\}$. Lemma 4 states that the expected height h_v is constant and that w.h.p. for all $v \in S$, $h_v \leq 2 \log \log n$:

Lemma 4. *Let $v \in S$, then $\mathrm{E}[h_v] = O(1)$, and moreover,*

$$\Pr[\bigwedge_{v \in S} h_v \leq 2 \log \log n] = 1 - O(\frac{1}{\sqrt{n}}),$$

where the probability is taken over the occurrence of faults.

This guarantees that the number of rounds is at most $2 \log \log n$. To show $O(\sqrt{n})$ probe complexity, we need to prove there is a concentration around $\mathrm{E}[h_v]$ w.h.p . The variables h_v are not independent since the trees may intersect, and therefore Chernoff inequality cannot be directly applied. This problem is addressed through dividing the random variables h_v into a logarithmic number of sets, such that within each set, all variables are independent.

Theorem 3. *The number of elements probed by the adaptive algorithm is w.h.p. $O(\sqrt{n})$. The number of rounds required in Step 2c of the algorithm is w.h.p. at most $2 \log \log n$.*

Early Stopping. We showed that a live quorum can be found in the And-Or quorum system when each processor fails with some constant probability p, in time $\theta(\log \log n)$ w.h.p . It is desirable that in a different fault model, for example when the failure probability of a processor is $o(1)$, the running time will shorten, namely, achieving early stopping of the algorithm. When no faults occur at all, the adaptive algorithm finds a live quorum in a single round. When the failure probability is $O(\frac{1}{n^c})$ for some constant $c > 0$, then w.h.p. no subtree generated in the adaptive algorithm grows to a height of more than a constant. It can be shown in this case that w.h.p. the number of steps required from the algorithm is a constant.

3 The And-Or Quorum System on a Balanced Binary Tree

Till now we have discussed the And-Or quorum system in the case the And-Or structure is defined over a complete binary tree. In Section 4 we present two ways of maintaining a binary tree over a dynamic environment of processors which is incomplete. In such a case the definition of the recursive structures, ANDset(*root*) and ORset(*root*) remains well defined. Lemma 2 can be extended to show that the intersection property is maintained, so the AndOr(*root*) structure is a quorum system. However, the load, availability and probe complexity of the new system are no longer guaranteed to be optimal as in the original And-Or

system. To maintain the optimal properties of the And-Or we restrict ourselves to balanced trees.

A family of trees is a set of trees $\{T_i\}_{i=1}^{\infty}$ such that T_i has i leaves.

Definition 2 (Balanced Tree). *A family $\{T_i\}_{i=1}^{\infty}$ of trees is said to be balanced if the following holds:*

- *Each node in the tree is either a leaf or has two children.*
- *There exists a constant $c \geq 0$ s.t. for each tree T_i, the difference between the levels of the leaves is bounded by c.*

Theorem 4 states that the And-Or on balanced trees maintains optimal asymptotic load and availability.

Theorem 4. *Let T be a tree in a balanced family of trees with n leaves. Let $\mathcal{Q} = \mathrm{AndOr}(T)$ be the And-Or quorum system defined on T. The load of $\mathcal{Q}$ is $\mathcal{L}(\mathcal{Q}) = O(\frac{1}{\sqrt{n}})$. Also, when the processors fail independently with probability $p < p_c$, for a constant p_c that depends only on the balance factor c, the global fail probability is:*

$$F_p(\mathrm{AndOr}(T)) \leq \exp(-O(\sqrt{n})).$$

4 The Dynamic And-Or Quorum System

In this section we present an adaptation of the Balanced And-Or quorum system over a dynamic network. We present two implementations: The first is on top of the Identity-Management scheme [12]. In this scheme a balanced binary tree structure is maintained over the active processors in the network and the embedding of the quorum system is straightforward. The second is on top of the *distance-halving* network, which emulates the De-Bruijn network in a dynamic environment (see [4,6] for other dynamic implementations of this network). In this network, communication lines correspond to edges of a binary tree, so that this implementation saves in the message complexity of finding a quorum.

For the entire section, we make the following simplifying assumption: An underlying layer takes care of concurrent operations of Join/Depart/'Use Quorum', so that the network implementation gets these operations sequentially (where 'Use Quorum' is an operation that sends a message to all the members of a quorum). We refer the reader to [9] where techniques for achieving concurrency in peer-to-peer networks are introduced.

4.1 A Dynamic And-Or Quorum System Using the Identity-Management Scheme

The first implementation of the And-Or quorum system in a dynamic environment uses the Identity-Management scheme presented by Manku [12]. In this scheme a balanced binary tree, with the active processors acting as its leaves, is maintained at all times, as processors join and depart from the system. The

binary tree structure defines the hosts' identities. Only leaf nodes of the tree correspond to a processor's identity. The internal nodes of the tree are conceptual. The sequence of 0s and 1s along the path from the root to a leaf node constitutes the identity of that leaf. The Identity-Management scheme keeps the tree structure balanced such that at each moment, all leaves are at one of three bottom levels of the tree. This balanced tree is used as a balanced And-Or tree, which was shown in Section 3 to generate a quorum system with optimal properties.

As was mentioned in Section 1.2, a dynamic environment raises two design issues of *integrity* and *locality*. To maintain *integrity* when the tree structure changes, all the quorums in use that are no longer legitimate quorums, must be modified. In the Identity-Management scheme there are two distributed algorithms that change the tree structure, one for joining a processor and one for processor departure. The algorithms for modifying quorums during join/depart are simple and are left for a full version of this paper.

To address the issue of locality we show how a processor can send messages to a random quorum without any knowledge of the tree structure. We use the fact that for each processor with identity x, level $|x| - 2$ is guaranteed to be complete. Details are left for a full version of this paper.

4.2 A Dynamic And-Or Quorum System over the Distance Halving Network

We now describe an adaptation of the balanced And-Or quorum system of Section 3 to a dynamic environment over the distance halving (DH) network of Naor and Wieder [14]. This implementation in contrast to the one of Section 4.1, strongly couples the quorum system with the dynamic network edges, saving in communication complexity of algorithms for selecting a quorum.

We use the same embedding of a binary tree over the DH-network, that we defined in [13]. We briefly repeat the description of the DH-network and of the binary tree embedding in it. The key idea behind most constructions of distributed hash tables, is to partition a (continuous) range of keys (denoted I) among the participating processors in the network. In our case I is the interval $[0, 1)$. Denote by $\boldsymbol{x}$ a set of n points in I. The point x_i serves as the identity of the ith processor, in a network of n processors. The points of $\boldsymbol{x}$ divide I into n segments. Define the segment of x_i to be $s(x_i) = [x_i, x_{i+1})$ $(i = 1, \ldots, n-1)$ and $s(x_n) = [x_n, 1) \cup [0, x_1)$. In the DH-network, each processor u_i is associated with the segment $s(x_i)$. If a point y is in $s(x_i)$ we say that u_i covers y. Notice that each point $y \in I$ is covered by exactly one processor. When data is written to y it is being stored in the processor that covers y, and remains stored in a single processor throughout the lifetime of the DH-network.

For a set $S \subset I$ and an identities vector $\overrightarrow{x}$, we denote by $\mathrm{CP}_{\overrightarrow{x}}(S)$ the set of processors that cover the points in S, i.e., $\mathrm{CP}_{\overrightarrow{x}}(S) \stackrel{\triangle}{=} \{p_i | \ \exists s \in S, s \in s(x_i)\}$. In case $\overrightarrow{x}$ is clear from context we omit it and simply write $\mathrm{CP}(S)$.

We recall the construction of the DH-network from [14]. We first define the DH-graph $G_c = (E_c, V_c)$. The vertex set of G_c is the unit interval I. The edge

set of G_c is defined by the following functions:

$$\ell(y) \stackrel{\triangle}{=} \frac{y}{2},\ r(y) \stackrel{\triangle}{=} \frac{y+1}{2},$$

where $y \in I$, and ℓ and r abbreviates 'left' and 'right' respectively. The edge set of G_c is then

$$E_c \stackrel{\triangle}{=} \{(y, \ell(y)), (y, r(y))|\ y \in I\}.$$

We denote by $parent(x)$ the function that returns the parent of a point $x \in I$ in the DH-graph i.e., $parent(x) = \begin{cases} 2x, & x < \frac{1}{2}, \\ 1-2x, & x \geq \frac{1}{2}. \end{cases}$ A pair of processors (u_i, u_j) are connected if there exists an edge (y, z) in the DH-graph, such that $y \in s(x_i)$ and $z \in s(x_j)$.

To implement the And-Or system on a DH-network we identify an embedding of a binary tree on the DH-graph. The $\ell(\cdot)$ and $r(\cdot)$ functions induce an embedding of a complete binary tree on every point x in I, in a natural way. The left child of a point x is $\ell(x)$ and the right child is $r(x)$.

An important parameter of the DH-network studied in [14] is the ratio between the largest and smallest cells induced by a composition. More formally:

Definition 3 (From [14]). *The* smoothness *of* $\overrightarrow{x}$ *denoted by* $\rho(x)$ *is defined to be* $\max_{i,j} \frac{|x_i|}{|x_j|}$. *If* $\rho(x)$ *is a constant independent of* n *we say that* $\overrightarrow{x}$ *is* smooth.

The smoothness of a network is maintained by the join/depart algorithms, in which processors choose their identities. Several methods for achieving smoothness are described in [7,10,12,1,14]. In the following dynamic adaptation of the And-Or system we assume both the network is kept smooth at all times, with a constant factor ρ, and that each processor has an approximation of $\log n$ (up to an additive factor). The Identity Management scheme [12] achieves a smoothness factor $\rho = 4$. In this scheme a node can also approximate the value of $\log n$ up to an additive constant of ± 2.

The Dynamic And-Or Quorum System Based on a Smooth Distance Halving Network. We first assume that at any moment, each processor has a local variable $l = \lceil \log n \rceil$, where n is the number of currently active processors in the network. Similar methods can be used when only an approximation of l within an additive constant is available.

The set of quorums is defined using an embedding of a binary tree T in the DH-graph, rooted in an arbitrary fixed point $r \in I$ with depth l. A set $S \in \text{ANDset}(r)$ is a set of points in I and so are sets in $\text{ORset}(r)$. Note that T has between n and $2n$ leaves. A quorum set is defined as the set of processors that cover a quorum (of points in I) in $\text{AndOr}(T)$, i.e.,

$$\mathcal{Q} = \{S \cup R|\ S = \text{CP}(S'), R = \text{CP}(R'), S' \in \text{ANDset}(r), R' \in \text{ORset}(r)\}.$$

Observation 2. *For any identities vector assignment $\overrightarrow{x}$, $\mathcal{Q}$ is a quorum system.*

When $\overrightarrow{x}$ is smooth each processor covers at most a constant number of leaves, which guarantees $\mathcal{Q}$ posses of an $O(\frac{1}{\sqrt{n}})$ load:

Theorem 5. *Let $\mathcal{Q}$ be the dynamic And-Or quorum system as defined above. If $\overrightarrow{x}$ is smooth then the load $\mathcal{L}(\mathcal{Q}) = O(\frac{1}{\sqrt{n}})$.*

Quorum Selection Through Gossip and Adaptation to Network Updates. The fact that the edges of the And-Or tree correspond to real communication lines of the DH-network enables a gossip based protocol which saves in message complexity. The dynamic And-Or presented in Section 4.1, requires that a message be sent to each of the processors selected. Let r denote the expected number of routing hops per message. r is typically logarithmic in the number of processors [1,14,20,21]. If a message is sent to m processors, the number of messages actually sent is blown to rm. In the And-Or system based on the DH-network, messages can percolate through the nodes that cover the tree in a similar manner to the recursive procedure for choosing a member of the ANDset collection (see Section 2). In this manner the total number of messages sent is less than $2\sqrt{n}$.

To keep a quorum consistent with the changing tree structure, it is necessary to modify the quorums in use as the network evolves. Each processor p, maintains a local variable $S(p)$ which is the set of all leaves (points in I), which are members of a quorum in use, and are covered by p. p updates $S(p)$ whenever one of the following events occur: First, when l, the local approximation of the height of the And-Or tree changes. Second, when the segment covered by p shrinks/grows, as a result of neighbor join/depart.

When l increases, each leaf x in the tree splits into $\ell(x), r(x)$. When l decreases, x becomes parent(x). p updates $S(p)$, by 'passing' all members $x \in S(p)$ to either $\ell(x)$ or $r(x)$ or parent(x) in a manner that resembles the gossip protocol. In the case where the segment covered by p changes, p either inserts new leaves into $S(p)$ or deletes leaves in $S(p)$ which are no longer covered by it, and passes them to the new processors that cover them. Note that each update to this distributed data structure, can be done by sending a messages via a single edge in the DH-network.

Analyzing the Availability and Probe Complexity of the Dynamic And-Or. In the dynamic And-Or environment we say a point $x \in I$ is alive if and only if the processor that covers it is alive. Otherwise it is said to be dead. A subset of points in I is said to be alive iff all its members are alive. Note that a live quorum of processors in the dynamic And-Or exists, iff a live quorum (of points) exists on the embedded And-Or structure.

In the And-Or system the analysis of F_p was kept relatively simple due to the fact that the failure probabilities of two disjoint subtrees are independent. In the dynamic case however, though processors fail independently, the failure of leaves in the tree is not independent. The status of two leaves is dependent only

if they are covered by the same processor. In a smooth network each processor covers at most a constant number of leaves in the embedded tree, so the event that a leaf fails depends on at most a constant number of other such events.

Due to this limited dependency, we can use the technique of domination by product measure which was introduced by Liggett et al.[8], and was used by Naor and Wieder in [15] in a similar context. The intuition behind this technique is that the above configuration is very similar to a configuration where each processor covers exactly one leaf, in which case leaves fail independently of each other and the availability analysis is as in the classic network.

The following theorem ensures high availability for the dynamic And-Or:

Theorem 6. *There exists a constant $p_c > 0$ such that for every $p \leq p_c$, the failure probability of the dynamic And-Or quorum system over a smooth DH-network is $F_p = \exp(-\theta(\sqrt{n}))$.*

We now turn to discuss probe complexity in the dynamic case. We use essentially the same algorithms described in Section 2, but here we first select a set of leaves and then use the set of processors that cover them. The analysis here is more complicated. The events that leaves in different subtrees contain live sets are no longer independent since two points in disjoint trees can be covered by the same processor.

We need to show that the non-adaptive algorithm applied to the dynamic case finds a live quorum w.h.p., in order to attain $O(\sqrt{n}\log n)$ probe complexity. The proof of Theorem 2 used the fact that for $p \leq \frac{1}{4}$, an And-Or quorum system on a subtree of height $2\log\log n$ has a failure probability $F_p \leq \frac{1}{n}$. Theorem 6 for the dynamic And-Or stated that for p small enough, the same applies.

Theorem 7. *There exists a constant $p_c > 0$ s.t., when the failure probability $p \leq p_c$, then the set of processors chosen by the non-adaptive algorithm (Section 2), contains a live quorum w.h.p . This set is of size $O(\sqrt{n}\log n)$.*

The adaptive algorithm for the dynamic case also works in the same way as in the classic case. Again, Theorem 6 can be used to show that the local subtrees generated in step 2c of Algorithm 1, have an expected constant height. Hence, on average the algorithm probes $\theta(\sqrt{n})$ processors. However, unlike in the classic case, here there is dependency between the height of different subtrees, so Chernoff bound cannot be used to bound the probability of large deviation. Following the Markov inequality, the probability for $\omega(\sqrt{n})$ probes cannot be a constant. Hence, with probability $1 - o(1)$, the number of probes required is $\theta(\sqrt{n})$. However, we did not show this happens with probability $1 - n^{-\alpha},\ \alpha > 0$. Therefore, bounding the probe complexity of the adaptive algorithm with a bound lower than $O(\sqrt{n}\log n)$ remains an open problem.

4.3 Comparing with Other Dynamic Quorum Constructions

Comparing with other dynamic constructions of quorum systems, the *dynamic And-Or* system holds some advantages. Most of these advantages were already

stated and are summarized in Table 1. A major advantage of the dynamic And-Or lays in the strong coupling between the combinatorial structure of the And-Or and the overlay network structure. For example, in the *Dynamic Paths*[15], whenever a processor joins or departs from the system a local computation of a Voronoi diagram is required. In the dynamic And-Or, join/depart events only require the insertion/deletion of a constant number of values to a set.

It was shown in [17] that for every quorum system, the global failure probability $F_p \overset{n\to\infty}{\to} 1$, when $p > \frac{1}{2}$. A 'critical failure probability' is the maximum p, for which the system is available i.e., for which F_p converges to 0 as n goes to infinity. It was shown in [15] that the critical probability for *'Dynamic Paths'* is $\frac{1}{2}$. The critical value for the And-Or system is $\alpha = \frac{3-\sqrt{5}}{2} < \frac{1}{2}$. The best failure probability p for which we have shown availability in the Dynamic And-Or, is a constant greater than 0, but strictly less than α. The actual critical value remains unknown, but in any case the 'Dynamic Paths' achieves a better result in this criteria.

Table 1. A comparison between the dynamic And-Or, the dynamic Paths and the dynamic probabilistic quorum system

	Dynamic And-Or	Dynamic Paths[15]	Dynamic PQS[2]
Load	$\theta(\frac{1}{\sqrt{n}})$	$\theta(\frac{1}{\sqrt{n}})$	$\theta(\frac{\sqrt{\log n}}{\sqrt{n}})$
Failure Probability	$e^{-\Omega(\sqrt{n})}$	$e^{-\Omega(\sqrt{n})}$	$e^{-\Omega(n)}$
Adaptive Algorithmic PC	$O(\sqrt{n})$	$O(\sqrt{n})$	$O(\sqrt{n}\log n)$
Number of rounds	$\theta(\log\log n)$	$\theta(\sqrt{n})$	1
Critical p	$0 < p_c \leq \frac{1}{4}$	$\frac{1}{2}$	-

5 Conclusions and Open Questions

We have shown that the And-Or quorum system has an adaptive algorithm for finding a live quorum, that operates in $\log\log n$ rounds and has a $O(\sqrt{n})$ probe complexity. Is there a quorum system with an optimal load for which there is an adaptive algorithm that works in a constant number of rounds (and the same probe complexity)? Recall that Naor and Wieder showed a lower bound on the probe complexity as a function of the load for non-adaptive algorithms (Theorem 1). This can be viewed as a lower bound on single round algorithms. A more general goal would be to extend this lower bound for adaptive algorithms with an arbitrary number of rounds. A milestone would be to prove the optimality of our algorithm for finding a live quorum in the And-Or system, or to find an algorithm with better probe complexity.

In Section 4 we showed that most of the good properties of the And-Or quorum system are maintained in the dynamic implementation over the DH-network. We showed that the algorithmic probe complexity of our adaptive algorithm is $O(\sqrt{n})$ on expectation and even with probability $1 - o(1)$. However, showing the probe complexity is $O(\sqrt{n})$ w.h.p. remains open as well.

Acknowledgements

We would like to thank Ilan Gronau for thoroughly reading this paper, and Udi Wieder and the anonymous referees for their useful comments.

References

1. I.Abraham, B.Awerbuch, Y.Azar, Y.Bartal, D.Malkhi, and E.Pavlov. A generic scheme for building overlay networks in adversarial scenarios. *IPDPS*, 2003.
2. I.Abraham ,D.Malkhi. Probabilistic quorums for dynamic systmes. *DISC*, 2003.
3. R. Bazzi. Planar quorums. *Distributed Algorithms, WDAG*, 1996.
4. P.Fraigniaud, P.Gauron. An overview of the content-addressable network d2b. *PODC*, 2003.
5. Y.Hassin, D.Peleg. Average probe complexity in quorum systems. *PODC* 2001.
6. M. F. Kaashoek, D. R. Karger. Koorde: A simple degree-optimal distributed hash table. *IPTPS*, 2003.
7. D. Karger, M. Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. *SPAA* 2004.
8. T. Liggett, R. Schonmann, and A.Stacey. Domination by product measures. *The Annals of Probability*, 25(1) 1997.
9. N.Lynch, D.Malkhi, and D.Ratajczak. Atomic data access in distributed hash tables. *IPTPS*, 2002.
10. D.Malkhi, M.Naor, and D.Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. *PODC*, 2002.
11. D.Malkhi, M.Reiter, and R.Wright. Probabilistic quorum systems. *PODC*, 1997.
12. G. S. Manku. Balanced binary trees for ID management and load balance in distributed hash tables. *PODC*, 2004.
13. U.Nadav, M.Naor. Fault-tolerant storage in a dynamic environment. *DISC*, 2004.
14. M.Naor, U.Wieder. Novel architectures for p2p applications: the continuous-discrete approach. *SPAA*, 2003.
15. M.Naor, U. Wieder. Scalable and dynamic quorum systems. *PODC*, 2003.
16. M.Naor, A.Wool. The load, capacity, and availability of quorum systems. *SIAM Journal on Computing*, 27(2), 1998.
17. D.Peleg, A.Wool. The availability of quorum systems. *Inf. Comput.*, 123(2), 1995.
18. D.Peleg , A.Wool. How to be an efficient snoop, or the probe complexity of quorum systems. *SIAM Journal on Discrete Mathematics*, 15(3), 2002.
19. S.Ratnasamy, P.Francis, M.Handley, R.Karp, and S.Shenker. A scalable content addressable network. *Proc ACM SIGCOMM*, 2001.
20. I.Stoica, R.Morris, D.Karger, F.Kaashoek, and H.Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. *ACM SIGCOMM Conference*, 2001.
21. B.Y. Zhao and J. Kubiatowicz. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB CSD 01-1141, University of California at Berkeley, 2001.